Fault Tolerance of Self Organizing Maps

Bernard Girau, César Torres-Huitzil

Noname manuscript No.
(will be inserted by the editor)

Fault Tolerance of Self Organizing Maps

Bernard Girau · Cesar Torres-Huitzil

Received: date / Accepted: date

Abstract Bio-inspired computing principles are considered as a source of promising paradigms for fault-tolerant computation. Among bio-inspired approaches, neural networks are potentially capable of absorbing some degrees of vulnerability based on their natural properties. This calls for attention, since beyond energy, the growing number of defects in physical substrates is now a major constraint that affects the design of computing devices. However, studies have shown that most neural networks cannot be considered intrinsically fault tolerant without a proper design. In this paper, the fault tolerance of Self Organizing Maps (SOMs) is investigated, considering implementations targeted onto field programmable gate arrays (FPGAs), where the bit-flip fault model is employed to inject faults in registers. Quantization and distortion measures are used to evaluate performance on synthetic datasets under different fault ratios. Three passive techniques intended to enhance fault tolerance of SOMs during training/learning are also considered in the evaluation. We also evaluate the influence of technological choices on fault tolerance: sequential or parallel implementation, weight storage policies. Experimental results are analyzed through the evolution of neural prototypes during learning and fault injection. We show that SOMs benefit from an already desirable property: graceful degradation. Moreover, depending on some technological choices, SOMs may become very fault tolerant, and their fault tolerance even improves when weights are stored in an individualized way in the implementation.

Keywords Fault tolerance · Self organizing maps · Hardware implementation · FPGA

B. Girau
Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
E-mail: bernard.girau@loria.fr

C. Torres-Huitzil
Cinvestav-Tamaulipas, Mexico
E-mail: ctorres@tamps.cinvestav.mx

1 Introduction

Artificial neural networks are computational models with a renewed research interest in artificial intelligence related applications. Additionally to a high performance in solving several tasks with generalization capabilities, artificial neural networks are generally assumed to acquire some other desirable features of their biological counterparts, such as their tolerance against imprecision, uncertainty and faults, which make them even harder to study or design [2]. The brain has a highly parallel information processing architecture made of seemingly imperfect and slow, but exceptionally adaptive and power-efficient components that carry out information processing functions, able to tolerate a small amount of synapse or neuron faults or even use noise as a source of computation [15][23]. Moreover, brains are able to relearn by growth of new neurons and/or neural connections and/or retraining of the existing neural architecture. Derived from these observations, the majority of neural network models are assumed to have built-in or intrinsic fault tolerance properties due to their parallel and distributed structure, and the fact that they contain more neurons or processing elements than the necessary to solve a given problem, i.e., some natural redundancy due to overprovisioning. Claiming an equivalent fault tolerance only on the basis of rough architectural similarities therefore cannot hold true in general, especially for small size neural networks [31]. As a matter of fact, studies have shown that most neural networks cannot be considered intrinsically fault tolerant without a proper design.

Obtaining truly fault tolerant neural networks is a very attractive and important issue both i) for artificial intelligence based solutions, where, for instance, pervasive embedded systems will require smart objects fully merged with the environment in which they are deployed to cope with unforeseeable conditions [32], and ii) as a source for reliable computing systems built from unreliable components, as suggested in [16]. Rooted on neural computing, a new paradigm is needed to take advantage of new emerging devices at nanoscale dimensions and deal with both manufacturing defects and transient faults. This trend might lead to even consider faults/errors as an essential/intrinsic part of a system design. In this last direction, the robustness and the potential fault-tolerant properties of neural models call for attention as permanent and transient faults, device variation, thermal issues, and aging will force designers to abandon current assumptions that transistors, wires, and other circuit elements will function perfectly over the entire lifetime of a computing system, relying mainly on digital integrated circuits [20]. Furthermore, this is particularly important for novel neural hardware accelerators and neuromorphic hardware design, which have attracted significant commercial efforts to produce more brain-inspired chips, such as the IBM TrueNorth chip.

Several experimental and less analytic works have been carried out to study neural networks fault tolerance, including the analysis of noise on the output sensitivity [11], the weight error sensitivity [35], and the relationship among fault tolerance, generalization and model complexity [2] [31] [27]. Most works have been focused on feedforward neural networks, at different levels of ab-

straction, from low level physical implementations to the high level intrinsic fault masking capacity of neural paradigms, see [28] for a more comprehensive review. However, few attempts have been made to study fault tolerance of other models, such as self-organizing maps (SOMs). SOMs are expected to tolerate some faults thanks to their self-organizing mechanisms, but their fault tolerance capabilities have not yet been fully explored. Herein, we consider that the principle of self-organization itself might have an influence onto the fault tolerance of computing systems and particularly of neural networks hardware implementations. This is why we chose to study SOMs because they stand as one of the most well-recognized models of self-organizing computing systems. Therefore, in this paper, we are not only interested in the way pre-learned SOMs react to faults, but also in the way fault tolerance evolves during the self-organization of the neural population.

Previously, in [34] authors studied the fault tolerance capability of SOMs in the presence of defective neurons undergoing stuck-at faults. It was shown that defective SOMs, in a linear array, can eventually re-organize themselves, by relearning, if the defective neuron stuck-output is larger than a critical value. Defective neurons were concentrated in one place in the array, forming what they called a defective-neuron cluster. In the experiments, 100 neurons, including six defective ones, were tested to show that the SOM can learn in spite of some faulty neurons. In [25], authors addressed fault tolerance improvement in SOMs by using a technique called fault immunization of the synaptic connections. Stuck-at-a faults, where a is a real value, were considered. Only one neuron was faulty at any time, but no restriction on the number of faulty links of the neuron was assumed. Weights are immunized by adding a constant value that is increased or decreased as much as possible without creating any misclassification. Fault immunization was formulated as an optimization problem on finding the corresponding constant value for each neuron. The application of this study was oriented to enhancing the reliability of the lossy image compression by a SOM. Other works are related to modifications of the Kohonen SOM on-line learning algorithm so as to improve the fault tolerance of the learned SOM. Section 3.3 describes more precisely these works.

Our present work is grounded on an extensive experimental study of the intrinsic fault tolerance of SOMs considering digital implementations on field programmable gate arrays (FPGAs), under bit-flip faults by analyzing their performance degradation with variable fault rates on synthetic datasets. FPGAs are suitable for neural networks hardware implementations because they combine high computing capability, logic resources and memory capacity in a single device. We are mostly interested in faults in weights, since when a SOM is intended to be implemented onto a digital hardware substrate, the value of each weight vector can be disturbed by various causes such as electromagnetic field or radiation, which can be modeled by a bit-flip fault model. This is particularly true for FPGAs devices, as it will be discussed in more detail later. Following our preliminary works reported in [29][10], this paper addresses multiple aspects in a combined way:

- We consider the evolution of this fault tolerance using different existing learning strategies to improve fault tolerance.
- In order to better analyze the effect of self-organization onto fault tolerance, we provide a behavioral analysis of the evolution of neural prototypes with faults using diverse datasets and SOM architectures.
- We also compare the fault tolerance of SOMs implemented in a distributed way to the case of a purely sequential hardware implementation. In this specific case, bit-flip faults affect multiple registers in addition to the stored weights.
- We also show how technological choices greatly influence the fault tolerance of SOMs, considering various weight storage policies: uniform storage resources (SWS: standard weight storage), individualized storage resources where register sizes are tuned to individual arithmetic precisions (IWS: individual weight storage), and limited individualized weight storage (OWS: optimal weight storage).

This very wide experimental study shows that SOMs intrinsically satisfy the very desirable property of graceful degradation. This property may even lead to a high level of fault tolerance, depending more on technological choices than algorithmic choices.

The paper is organized as follows. Section 2 introduces some concepts and briefly describes the general approach for fault tolerance assessment in neural networks. Section 3 presents the fundamentals of the Kohonen SOM model and the on-line learning mechanism for self-organization. Additionally, three techniques that have been used to improve the fault tolerance of SOMs, which involve a modification of the learning/training mechanism, are presented. Section 4 presents the hardware implementation choices for SOMs targeted to a fully parallel digital hardware implementation and a baseline sequential one, and the specificities for SOMs fault tolerance assessment. Section 5 describes the datasets used for testing and the obtained experimental results for different scenarios and weight storage policies.

2 Fault tolerance in neural models

Fault tolerance is a system attribute that makes it able to preserve its expected behavior after faults have manifested themselves within the system [4]. More precisely, a fault-tolerant system might be defined as one that has provisions to avoid failure after faults have caused errors within the system. When a statement about neural networks fault tolerance is made, it should be implicitly assumed a failure condition or criterion. That is, the threshold below which it cannot longer perform its function according to the specification. Thus, fault tolerance in neural networks depends on the definition of the acceptable degree of performance and its intended application [19].

At a high level of abstraction, neural networks fault tolerance can be analyzed by the effects of errors in the main operators, rather independent from their intended physical implementation. In a more comprehensive approach

[17], after this initial step, physical faults affecting a specific implementation can be mapped onto such errors to estimate fault tolerance of a given neural model, and by identifying critical components, complementary and ad-hoc fault tolerance policies can be further applied to enhance the properties of the neural model hardware implementation. In order to study fault tolerance, researchers develop models of them to examine the variety of faults that need to be tolerated during the operation of a given system.

2.1 Faults and fault models

The following fault models have been widely used as abstractions of physical defect mechanisms in digital electronics devices/systems [1], and also applied to neural networks hardware implementations:

- **Stuck-at**, a data or control line appears to be held exclusively high (stuck-at-1) or low (stuck-at-0).
- **Random bit flips**, a data or memory element has some incorrect, but random value.

The stuck-at fault model is very popular since many defects at the transistor and interconnection structures can be modeled, as permanent faults, at the logic level with reasonable accuracy. This model is a binary model that does not capture indeterminate states that faults may induce while occurring in communication channels and arithmetic-logic operators. The random bit-flips model transient faults that usually happen at registers or memory elements due to external perturbations, for instance, a single event upset (SEU), mainly caused when highly energetic particles strike sensitive regions of a circuit. Under this model, damage/corruption is done only to the data and not to the physical circuit itself. Conceptually, it consists of a register bit that is switched randomly, resulting in a memory element that holds a wrong logic value.

In SRAM-based FPGAs, the most common faults/errors are caused by SEUs that change the configuration and/or the user memory (flip-flops and block memory). Data associated with routing resources, LUTs, control signals, and the contents of smaller memory modules (e.g. flip-flops) are stored in configuration bits (CRAMs), and larger block memory modules are stored in block memory bits (BRAMs), which can be affected by SEUs. A bit flip, or SEU, in BRAMs, might bring the system to a transient failure (it is likely to be overwritten at the next clock cycle), but they may have long term effects if the flipping is in the CRAM, leading to an error in the logic function that persists until the configuration is refreshed. A LUT affected by a SEU in its CRAM will produce an incorrect value only when the input pattern is the one associated with the faulty configuration bit, while for every other input pattern the faulty LUT will behave correctly. This behavior cannot be simulated with the stuck-at model. Studying fault tolerance of neural networks on FPGAs should thus mostly emphasize error models related to very short-term bit-flips at the level of weight coding, and to more long-term conditional bit-flips at a computational level.

2.2 Fault injection

In order to evaluate the fault tolerance of a neural model, faults are probabilistically introduced into this model and the degree of failure, impact on the performed task, is evaluated according to some measures, which basically measure the performance distance (closeness) in performing a task by a fault-free neural network and the derived faulty networks [12]. The measure of fault tolerance from many experiments is evaluated against the number of faults injected into the neural model. The limit of the network fault tolerance is problem-dependent and determined by operating scenarios of multiple faults leading to a violation of the performance constraints. Exhaustive testing of all possible single faults is prohibitive, not to mention that multiple faults may occur concurrently. Hence, the strategy of randomly testing a small fraction of the total number of possible faults in a network has been adopted for tractability. It yields partial fault tolerance estimates that are statistically very close to those obtained by exhaustive testing.

3 Self Organizing Maps

3.1 Fundamentals

A SOM is a neural network able to perform data quantization while preserving some predefined topological relations among the set of learned prototypes. This model is a recognized tool for data visualization that has also been used for many practical applications such as pattern classification, image processing, and robotics. SOMs, as proposed by Kohonen [13], consist of neighboring neurons commonly organized on one- or two- dimensional arrays that project patterns of arbitrary dimensionality onto a lower dimensional array of neurons, where $\mathbf{r}_i$ denotes the coordinates of neuron i in this array. All neurons receive the same input pattern and an iterative mechanism updates the neuron's weights so as to learn to quantize the input space in an unsupervised way. This mechanism first selects the neuron whose weights best fit the given input pattern, and then brings the weights of this neuron and of its neighbors slightly closer to the current input pattern.

More precisely, each neuron in a SOM is represented by a d-dimensional weight vector, $\mathbf{m} \in \mathbb{R}^d$, also known as prototype vector, $\mathbf{m} = [m_1, ..., m_d]$, where d is the dimension of the input vectors, $\mathbf{x}$. Neurons are connected to adjacent ones by a neighborhood relationship, which defines the structure of the map. The mechanism for selecting the winning neuron requires a centralized entity, so that the Kohonen SOM is not a fully distributed model as the cortex organization [21]. After learning, or self-organization, two vectors that are close in the input space will be likely to be represented by prototypes of the same or of neighboring neurons on the neural map. The learned prototypes become ordered by the structure of the map, since neighboring neurons have similar weight vectors. In the literature, two main types of SOM can be dis-

tinguished, considering the number of neurons in the map with respect to the number of expected clusters: SOMs in which the number of neurons is roughly equal to the number of expected clusters to be found in the input, and SOMs with a large number of neurons (thousands or tens of thousands), which are used to allow the emergence of intrinsic structural features of the data space; these SOMs are referred to as Emergent Self-Organizing Maps (ESOMs) [30].

3.2 On-line mode training

The basic version of SOM learning is an on-line stochastic process, inspired by neurobiological learning paradigms. Several other extensions of the algorithm have been proposed [9]. In the SOM on-line mode algorithm, learning starts with an appropriate (usually random) initialization of the weight vectors, $\mathbf{m}_i$. The input vectors are presented to the SOM in multiple iterations. For each iteration, i.e. for each input vector $\mathbf{x}$, the distance from $\mathbf{x}$ to all the weight vectors is calculated using some distance measure. The neuron whose weight vector gives the smallest distance to the input vector $\mathbf{x}$ is called the best matching unit (BMU), denoted by c, and determined according to:

$$\|\mathbf{x} - \mathbf{m}_c\| = \min_i \|\mathbf{x} - \mathbf{m}_i\| \tag{1}$$

where $\|\cdot\|$ is the distance measure, typically the Euclidean distance, $\mathbf{x}$ is the input vector and $\mathbf{m}_i$ is the weight vector of neuron i. The winner c and its neighboring neurons $i \in N_w$ update their weights according to the SOM rule:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t) h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] \tag{2}$$

where t denotes the time, $\mathbf{x}(t)$ is an input vector randomly drawn from the input data set at time t, $\alpha(t)$ the learning rate at time t, and $h_{ci}(t)$ is the neighborhood kernel around c. The learning rate $\alpha(t)$ defines the strength of the adaptation, which is application-dependent. Commonly $\alpha(t) < 1$ is constant or a decreasing scalar function of t.

The neighboring kernel $h_{ci}(t)$, which is a function of the distance between the winner neuron c and neuron i, can be computed using a Gaussian function:

$$h_{ci}(t) = \exp\left(-\frac{\|\mathbf{r}_c - \mathbf{r}_i\|^2}{2\sigma^2(t)}\right) \tag{3}$$

Where $\|\mathbf{r}_c - \mathbf{r}_i\|$ is the distance between neuron i and the BMU c. The value of $\sigma(t)$ is fairly large in the beginning of the process, for instance in the order of 20% of the longer side of the SOM array, after which it is gradually reduced to a small fraction of it, e.g. 5% of the shorter side of the array [13].

An iterative batch version of SOM learning exists. Instead of using a single input vector at a time, the whole dataset (batch) is presented to the map before updating any weight [14]. In each training step, the dataset is partitioned according to the Voronoi regions of the map weight vectors, i.e., each data

vector belongs to the dataset of the closest map unit. This algorithm is deterministic, and one of its advantages is that the limit states of the prototypes depend only on the initial choices. However, in this paper, we are interested in the way fault tolerance evolves with SOM learning. The on-line mode training provides us with a more detailed evolution to observe. Moreover, we not only consider the standard Kohonen SOM learning algorithm, but also some variants among which several ones cannot be applied in batch mode. Therefore, we will only focus on the on-line mode training.

3.3 Fault tolerant training

This section briefly introduces three techniques for SOM that have been proposed to improve its fault tolerance: weight restriction, fault insertion, and noise injection during training.

3.3.1 Restricting weight values

This technique restricts the weight magnitudes to be low during training to potentially make the SOM more fault-tolerant [8][7]. For instance, by i) penalizing high magnitude weights in the learning rule; $m_i^j(t+1) \in [m_{min}, m_{max}]$, the lower and upper bound of the weight vectors components, respectively, or ii) balancing weights to distribute the weight values more uniformly, e.g., to distribute the absolute weight values around the average absolute value. Here, we use the thresholding method, in which a weight is updated only if its absolute value, computed by the learning rule, does not exceed the following threshold:

$$\theta(t) = \frac{1}{nw^2} \sum_{\mathbf{m}^j} \sum_{i=1}^{n} \left| m_i^j(t) \right|$$

3.3.2 Inserting faults

This technique randomly inserts faults during training [24] so as to improve fault tolerance. In each iteration, a fixed number of weights (neurons) are randomly chosen, then faults (bit-flips) are injected according to these steps:

1. Randomly choose a predefined number of weights and insert faults in them
2. Apply the learning rule of on-line (resp. batch) training for a pattern (resp. all patterns) in the training set
3. Restore faulty weights to their non-faulty state and repeat the process until a stop criterion is reached

In our tests, only one faulty bit is introduced (and then restored) for each learning iteration.

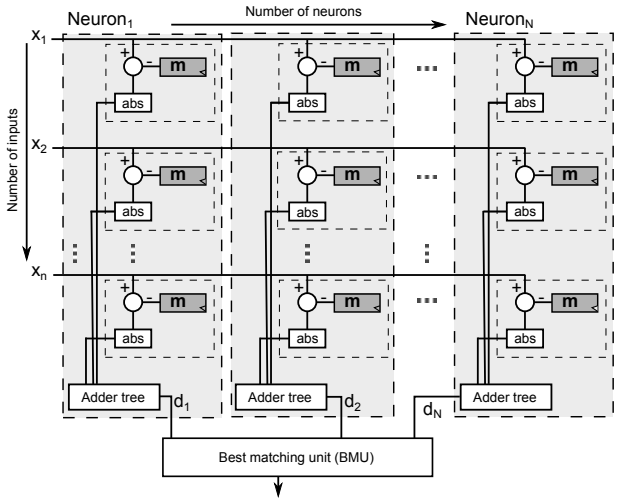


Fig. 1 Generic hardware architecture for a two-dimensional SOM.

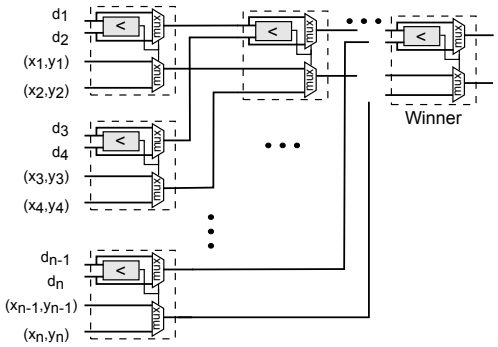


Fig. 2 BMU block diagram for a two-dimensional SOM.

3.3.3 Noise and weight perturbations

During training, in each iteration, a small number of inputs/weights are selected randomly to be perturbed by adding some type of noise [25][3]. The basic idea is to add noise to the inputs or weights of each neuron. Two main options exist, multiplicative and additive noise; additive noise being the most used. We consider such an additive noise in our tests.

4 SOM hardware implementation and fault tolerance assessment

4.1 Parallel implementation

We analyze SOM fault tolerance for a fully parallel digital hardware implementation of a map, taking only into account the SOM recall phase with a general

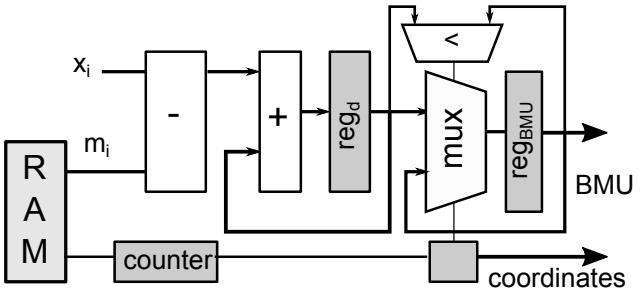


Fig. 3 Baseline sequential hardware implementation for a two-dimensional SOM.

architecture as shown in figure 1, since most embedded implementations of SOMs do not require on-chip learning. A b-bit fixed-point representation for the internal computations and storage is used, as most SOMs hardware implementations use this number system so as to optimize hardware resources. A neuron essentially consists of registers (to store weights), adders and absolute computation modules that as a whole compute the Manhattan distance. The original SOM uses the Euclidean distance to measure the distance between input patterns and weight vectors, but it requires hardware-greedy multiplications and square roots, thus the Manhattan distance is a preferable choice so that even the distance between the inputs $\mathbf{x}$ and the vector prototypes $\mathbf{m}$ can be computed concurrently at every clock cycle.

The BMU can be implemented as a comparator binary tree and a set of multiplexers, which receive the Manhattan distances and spatial coordinates from all neurons, as shown in figure 2. The BMU's outputs are the spatial coordinates (x, y) of the winner neuron. The neighborhood function plays a key role in SOM learning, but the direct implementation of the exponential function is not practical due to its complexity, instead look-up tables to store precomputed values can be used for resource saving. However, herein we only consider the on-chip implementation of the recall phase and an off-chip implementation for the learning phase, so that we do not need to compute the neighborhood function on-chip.

4.2 Sequential implementation

In the sequential version, the SOM is implemented as shown in figure 3, where a single neuron is time-multiplexed. We assume that enough memories (Block-RAMs in FPGAs) are available and organized into banks to read weights of a neuron in a single clock cycle. A counter generates the addresses to this memory bank to retrieve the weights for different neurons. Once weights are retrieved, the distance is computed from a given input and temporally stored in reg_d. On each clock, the content of this register is compared with the content of register reg_{BMU} so as to sequentially determine the minimum, which is

stored once again in reg_{BMU}. This process is repeated until all neurons in the SOM are processed. The control logic is not shown in figure 3 for simplicity.

4.3 SOM fault tolerance assessment

We consider that faults occur only in weight registers in the parallel hardware architecture shown in figure 1. This choice considers plausible hardware faults and not only the unprecise notion of faulty neuron, yet all possible hardware faults are not considered this way (see 2.1). More precisely, we assume that each bit in defective neuron weights, which are stored in b-bit registers, is flipped with some probability and remains the same during the SOM processing. The fault rate u is defined as the percentage of flipped bits (b_{flips} such bits) in neuron weights with respect to the total number of bits in the SOM map. It is used so as to evaluate fault tolerance in SOMs. Considering a $w \times w$ SOM map, with N inputs and b-bit wordlength for weights, yields:

$$u = \frac{b_{flips}}{w^2 \times N \times b} \tag{4}$$

It must be pointed out that the number of faulty bits and the rate of faulty neurons increase very rapidly with the above-defined fault rate: to give an example, if $b = 16$, $N = 4$, and $u = 0.05$, 5120 faulty bits appear (out of 102 400) in a 40×40 SOM (more precisely, for this case, an ESOM), and 96 % of all neurons contain at least one faulty weight. For space consideration, we have focused on such fault rates as they represent a more devastating effect on the SOM behavior than single, double or triple faults, which can be well masked in SOMs as it has been reported in related works [34][25], and confirmed by our experiments. Moreover, this fault model takes into account the fact that all the introduced faults are not equivalent: faults on least significant bits do not have the same effect as faults on most significant bits.

In the case of the sequential implementation, weights are stored in Block-RAMS. We assume that such data, as well as configuration bits, may be protected by means of specific techniques such as memory scrubbing [26], complementarily to the fault tolerance of the neural model, thus avoiding an accumulation of faults after several clock cycles. Therefore, we take into account only bit-flip faults that can occur during each clock cycle in the weights of the currently handled neuron, or in the counter of neurons, or in reg_d, or in reg_{BMU}, or in registers that store the coordinates (position) of the BMU.

4.4 SOM Performance measure

The severity of faults in the SOM can be characterized with respect to the fault rate u: it consists in considering the artificially introduced faults at a given fault rate as the independent variable, and a selected performance measure of the SOM as the dependent variable. The average quantization error (AQE) and

distortion measure are used as SOM performance criteria for fault tolerance, which provides a combined way to evaluate the result of the SOM convergence on a given dataset [6][18][5].

The AQE measures the representation fidelity of the training data, and it is computed as the average distance between each data vector and the prototype of its BMU that should be very close:

$$AQE = \frac{1}{n}\sum_{1}^{n}\left\|\mathbf{x}_i - \mathbf{m}^{c(\mathbf{x}_i)}\right\|^2 \tag{5}$$

where n is the number of input vectors, $\mathbf{x}_i$ is an input, $\mathbf{m}^{c(\mathbf{x}_i)}$ is the weight of the BMU in the map for $\mathbf{x}_i$. The less the AQE, the better the fidelity of the representation of input data.

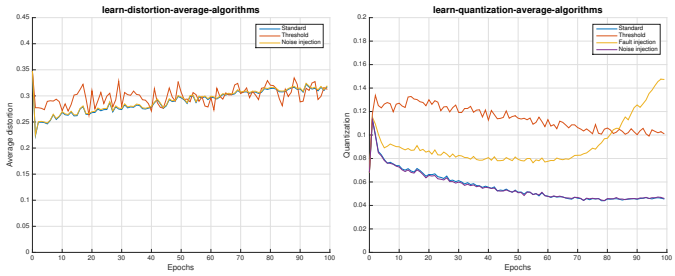
The distortion measure considers both vector quantization and structure preservation of the SOM [22] and it is suitable for comparing maps of the same size [18]. Let's consider a SOM represented by prototypes $\mathbf{m}^j$, the distortion measure is defined as the average over all input patterns (a set of n input data vectors $\mathbf{x}_i$) of the weighted sums of the squared distances between the pattern and all neuron prototypes, with weights that depend on the neighborhood function between each neuron and the winner neuron (BMU) for the given pattern. According to [33], the original distortion measure would be biased towards the 2D SOM. If the input's distortion is redefined to be the average distortion over its BMU's neighborhood, the results will not be affected by the different topologies of the grid and the distortion measure becomes:

$$\xi = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{m} h_{c(\mathbf{x}_i)j}\left\|\mathbf{x}_i - \mathbf{m}^j\right\|^2}{\sum_{j=1}^{m} h_{c(\mathbf{x}_i)j}} \tag{6}$$

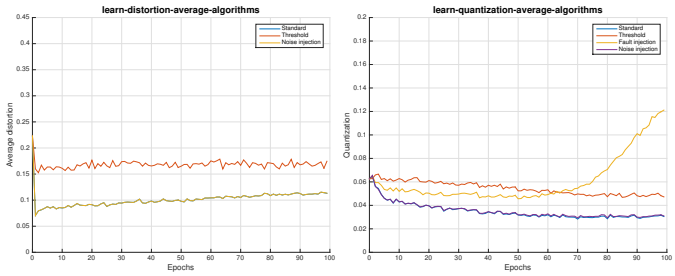
where $c(\mathbf{x}_i)$ is the BMU of $\mathbf{x}_i$, and $h_{c(\mathbf{x}_i)j}$ is the neighborhood function of neurons $c(\mathbf{x}_i)$ and j.

5 Experimental results

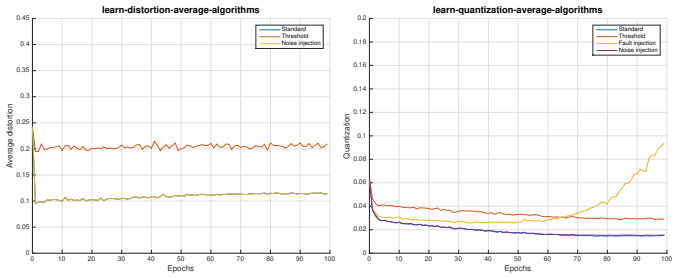
We have developed a software environment for fault tolerance evaluation, which handles computations using fixed-point representations (even during learning) for the different SOM hardware implementations. Experiments are performed with 4×4 up to 40×40 2D SOMs, whose weights are initialized to random values, evenly distributed between $[0, 1]$ and then scaled to a fixed-point representation Q6.10. Hereafter, we call this arithmetic representation standard weight storage (SWS). Artificial 2D and 3D datasets are used, with new points generated at each learning iteration according to the chosen distributions, instead of pre-generating fixed-size datasets, so as to get rid of any cross validation requirement. Studied distributions are: uniform distribution, mixtures of gaussians, or mixtures of uniform distributions within compact "boxes". All SOMs use the same learning rate, update radius and Gaussian



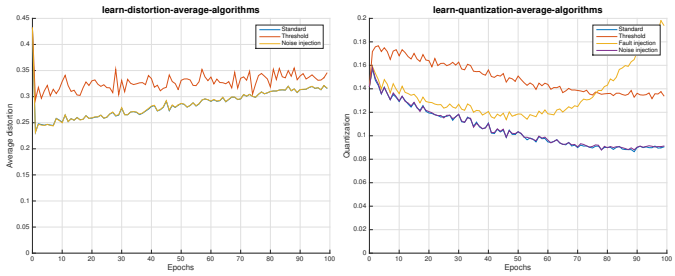
(a) 2D uniform distribution (Distortion, Quantization)



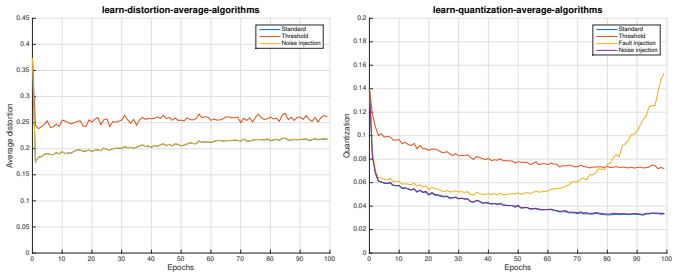
(b) 2D Gaussian distribution (Distortion, Quantization)



(c) 2D boxes distribution (Distortion, Quantization)



(d) 3D Gaussian distribution (Distortion, Quantization)



(e) 3D boxes distribution (Distortion, Quantization)

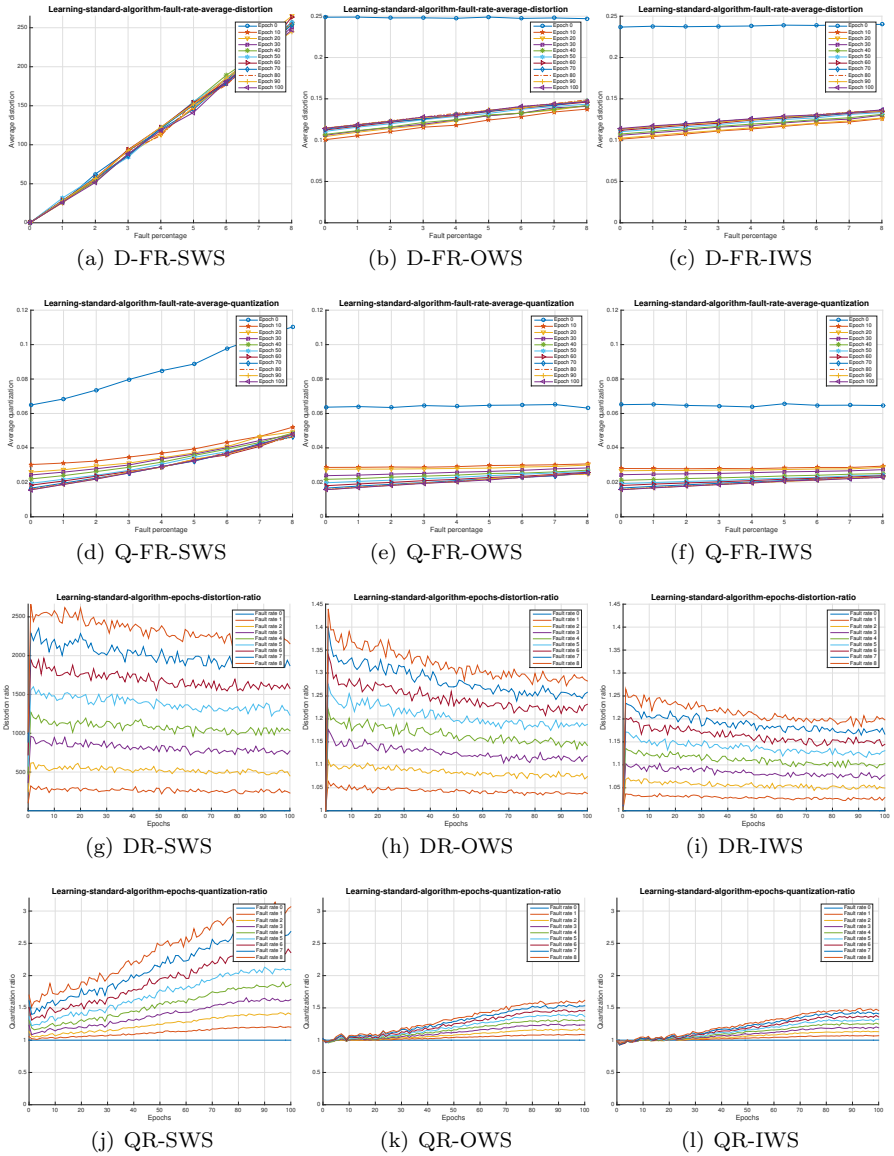
Fig. 4 Distortion and quantization for an 8×8 SOM during learning for different distributions and learning algorithms.

neighborhood function, during 100 epochs with 50 iterations each. Different map sizes and initializations are tested. For each iteration, various learning patterns are randomly chosen, and all learning techniques, map sizes and map initializations concurrently train with respect to each of these input patterns. Then their fault tolerance is estimated by generating several randomly faulty versions of each map at each epoch of each of the learning techniques that are tested, for each weight storage policy (see 5.2). These nested tests have resulted in a total of 10^8 simulations of faulty SOMs. For space considerations, most results are presented for 8×8 SOMs on 2D boxes distributions, with some exceptions when appropriated, clearly indicated. Several results for other distributions and map sizes were presented in our previous works [29] and [10]. Our observations are similar for all distributions and map sizes.

5.1 Fault tolerance evolution during learning

The quantization and distortion measure profiles during learning are shown in figure 4 for different 2D and 3D distributions, using different learning techniques. The presented results are averaged over several randomly initialized maps, but all learning techniques and maps use the same series of randomly generated input patterns. It explains why the different curves have similar irregular profiles. Both measures decrease during training with the standard online algorithm for all SOMs, but with different convergence rates for different distributions. This can be explained by the usual problems known for Kohonen SOMs that unfold themselves in a uniform distribution-like input space: some maps spread well while some others remain partially twisted.

Figure 4 shows that results for the standard algorithm mostly overlap with those produced by noise injection and are the best among the other techniques. Training with thresholding (to restrict weight values) yields larger quantization and distortion values, and a slower convergence rate. These results tend to show that this technique does not well extend to the kind of fault model we study here. Note that for thresholding, quantization starts to increase rapidly around epoch 60 during learning. This is even more noticeable for the fault injection based technique: its results are not even included in figure 4, for readability, since this method performs worst among the implemented algorithms. This is directly linked to the number representation system that we use herein, where faulty bits can be injected in the 6-bit integer part and induce excessively large weights even after bit restoration (e.g. 000000 becomes 100000 with a faulty bit, then 011111 with learning, and recovering the initial value of the faulty bit is useless here). Using different integer part sizes induces more or less bad results for this method. A similar problem appears when we inject faults in the SOMs to evaluate their fault tolerance. We address this issue in the next section by considering two different policies or variants for arithmetic representation.



(a) D-FR-SWS (b) D-FR-OWS (c) D-FR-IWS

(d) Q-FR-SWS (e) Q-FR-OWS (f) Q-FR-IWS

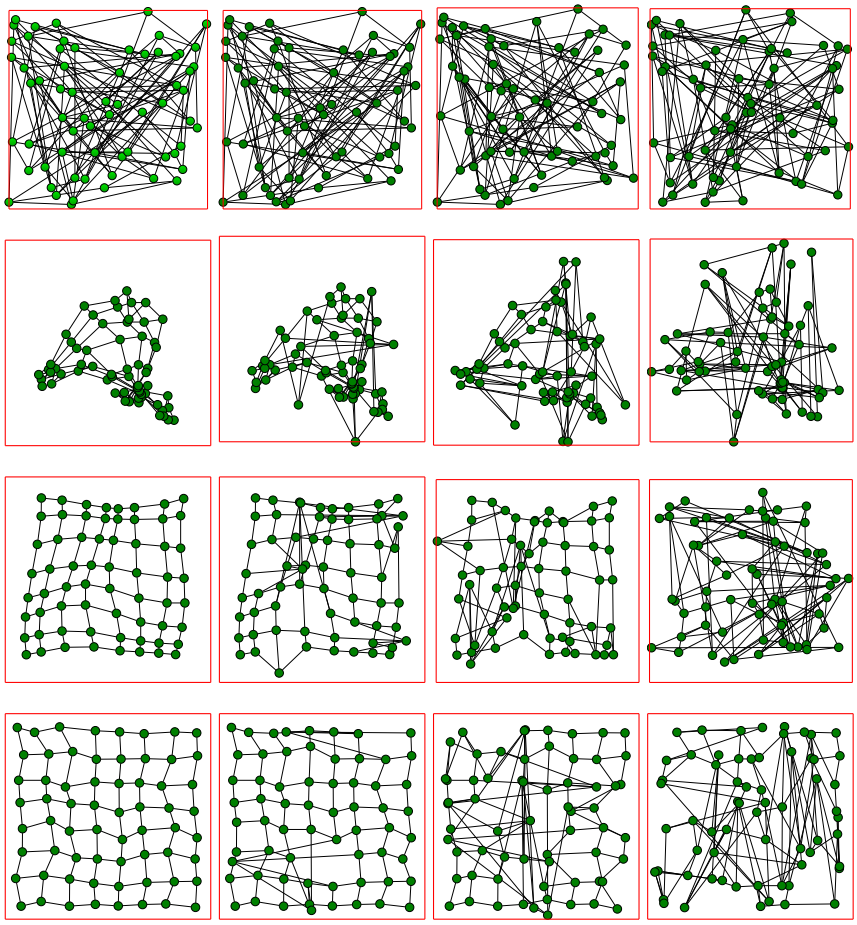
(g) DR-SWS (h) DR-OWS (i) DR-IWS

(j) QR-SWS (k) QR-OWS (l) QR-IWS

Fig. 5 Distortion/Quantization for different implementation choices (weight storage) on a 2D boxes distribution (8x8 map). (a-c): distortion vs fault rate for different epochs, (d-f): quantization vs fault rate for different epochs, (g-i): distortion ratio vs epochs for different fault rates, (j-l): quantization ratio vs epochs for different fault rates.

5.2 Optimal and individual weight storage

Herein, we consider the same general parallel hardware architecture as shown in figure 1, but with two different arithmetic precision choices, which can be



(a) Fault rate 0 % (b) Fault rate 1 % (c) Fault rate 2 % (d) Fault rate 9 %

Fig. 6 Evolution of prototypes during learning (epochs 0, 1, 20, 60 from top to bottom) with different fault rates for an 8×8 SOM with 2D uniform distribution using an OWS policy.

considered as complementary ad-hoc policies that can be further applied to enhance the fault tolerance properties of a specific neural model hardware implementation, i.e., SOMs for this study.

First, we consider an optimal weight storage (OWS) policy, where the only difference with respect to SWS is that the weights (coordinates of neural prototypes) are stored in a heterogeneous way: instead of using an homogeneous Q6.10 representation, register sizes are tuned individually to the minimal size that fits the significant bits of the integer part, thus resulting in a variable Q0.10 to Q6.10 representation of weights. Faults are introduced both to the significant integer part and the fractional part.

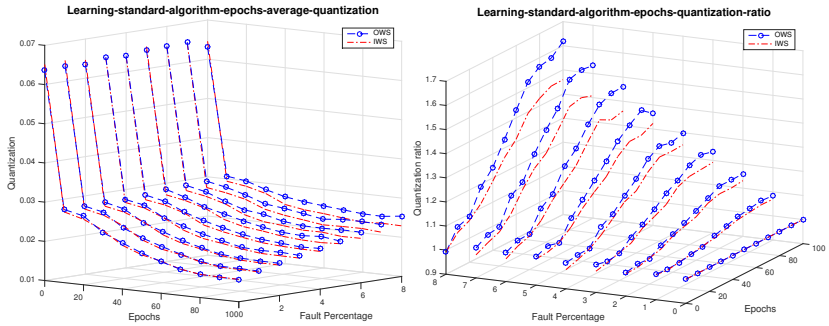
Second, an individual weight storage (IWS) policy is used, where once again, a heterogeneous fixed point representation similar to OWS is considered, but for this policy faults are limited only to the significant bits of each weight. For example, in a weight equal to 0.0001011001, bit flips will occur only in the 7 least significant bits of the fractional part. For this experiment quantization and distortion measures are evaluated under the same conditions as SWS and OWS.

Figures 5(a-f) show the distortion and quantization measures versus the bit-flip percentage introduced into SOMs for different weight storage policies. Recall that a 1% fault rate already corresponds to 5120 faulty bits in the largest SOMs we consider herein. As the same results are not expected for different faulty SOMs, even on the same dataset, for each fault rate several faulty versions of the trained SOMs were randomly generated and averaged in the presented results.

Distortion increases with the fault rate for all the cases (see figures 5(a)-(c)): however, for OWS and IWS distortion only slightly increases with injected faults but it rapidly increases even with a small percentage of faults for the SWS policy. OWS and IWS distortion results are much better (approximately three orders of magnitude: for example, 0.14 with OWS/IWS compared to 250 for SWS for an 8% fault rate and an 8×8 map). These results show that the structure perturbations have been reduced by the OWS/IWS policies that keep the faulty prototypes not too far from their initial position. When looking at the evolution of the neural prototypes across the map (see figure 6 for an 8x8 map on a 2D uniform distribution using the OWS policy when injecting faults), we understand that the initial distortion value corresponds to random prototypes, then it falls down during the first iterations of learning since the map 'gathers' its prototypes before unfolding in the input space, thus increasing again its distortion. Nevertheless figure 6 shows that significant fault percentages quite highly perturb the SOM structure that emerges from learning. The distortion measure appears as unable to highlight this fact, knowing that random prototypes provide a distortion that is not much higher than for a perfect grid.

Quantization also increases with the fault percentage below 10% for all weight storage policies (see figures 5(d)-(f)). However, when an error measure that considers the local behavior of the map is used, results clearly show that the structure preservation is affected even at small fault rate percentages, as the distortion measure indicates, especially for SWS policy. This suggests that faults in weights can be better masked in the SOM for OWS and IWS, the global internal representation being distributed enough to cover the input space even with faults. Nevertheless, all these results confirm that SOMs may stand as quite fault-tolerant models, since any abrupt degradation of their behavior with faults is observed.

Figures 5(g)-(l) illustrate in more detail how fault tolerance measures evolve through learning for different fault rates for stochastic online learning and for the different storage policies SWS, OWS and IWS. Herein, the level of fault tolerance is estimated as the *ratio* between the average (over all



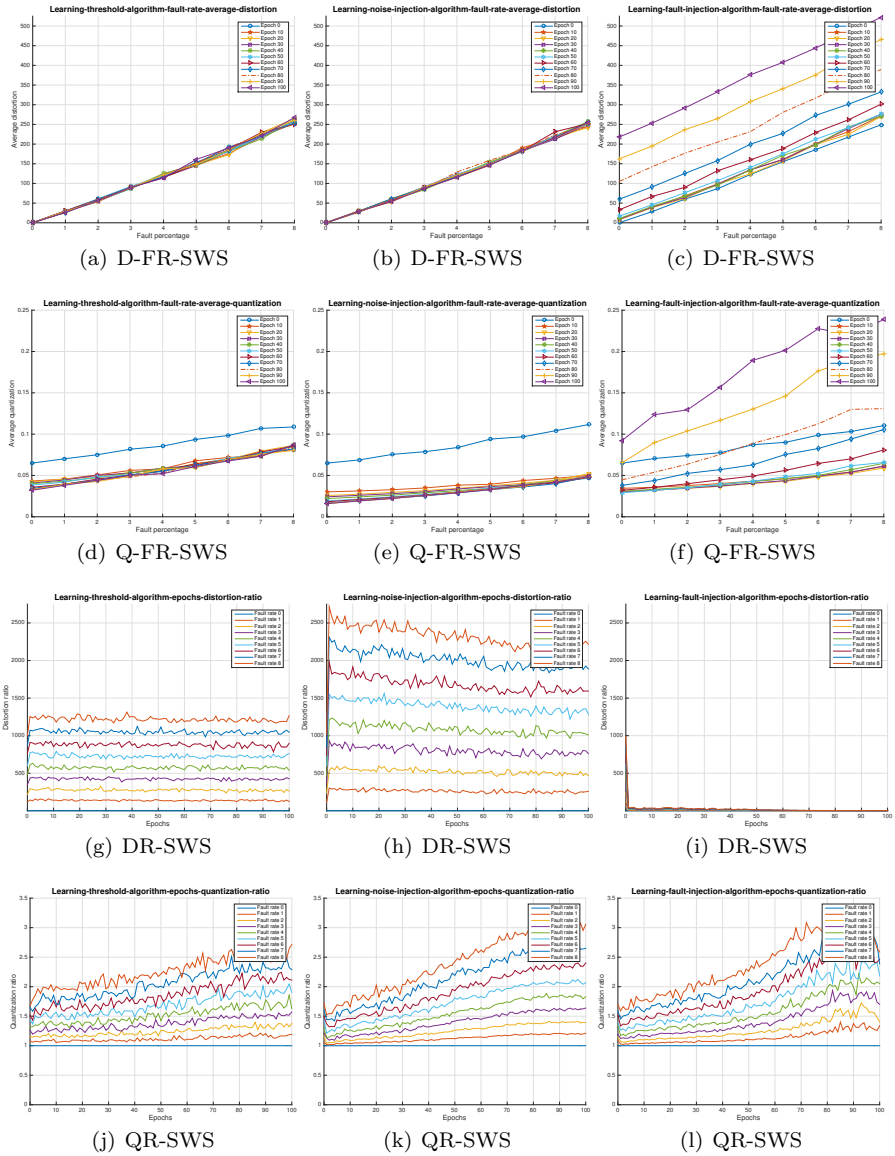
(a) Q-FR-OWS-IWS (b) QR–FR-OWS-IWS

Fig. 7 Comparing quantization for OWS and IWS on a 2D boxes distribution (8x8 map) for different fault rates.

tested faulty versions of each map) performance obtained for the given fault rate and the performance without faults for the corresponding epoch. Thus the 'optimal' or 'reference' value for this ratio is 1 (fault rate 0), and a ratio increase means a degradation of the SOM fault tolerance. With OWS and IWS, after an slight decrease of this ratio from the initialization to epoch 1, we see that it increases quite slowly for the quantization measure, and even stabilizes during the last epochs at a value that depends on the fault rate. For the distortion measure, there is an abrupt increase of the fault tolerance ratio up to very large values with SWS. Such increase is a direct consequence of the introduction of faulty bits in the integer parts of the weights. The order of magnitude of the ratio goes up to 10^3 with such faults in SWS, since the distortion measure takes into account the weights of all neurons, whereas the quantization measure only considers the winner neuron. Again, it is tightly linked to the size of the integer part, e.g. a Q4.12 instead of Q6.10 fixed point representation would induce approximately 16 times lower values for the maximum distortions with SWS. This negative effect on distortion is observed even when introducing only a few faulty bits in our other experiments, whereas the quantization-based fault tolerance remains optimal for such scarce faults. OWS and IWS greatly improve the distortion ratio, that even decreases during learning after the initial increase from epoch 0 to 1. IWS performs best, except for the quantization of uniform distributions, where OWS spreads weights more interestingly in the input space.

5.3 OWS vs IWS results

Figure 7 shows a closer view of how quantization and quantization ratio evolve during learning for OWS and IWS policies. Note that IWS is similar to OWS mostly for epoch 0 where prototypes are completely random and OWS faults are equivalent to noise. IWS policy is more fault tolerant than OWS not only



(a) D-FR-SWS (b) D-FR-SWS (c) D-FR-SWS

(d) Q-FR-SWS (e) Q-FR-SWS (f) Q-FR-SWS

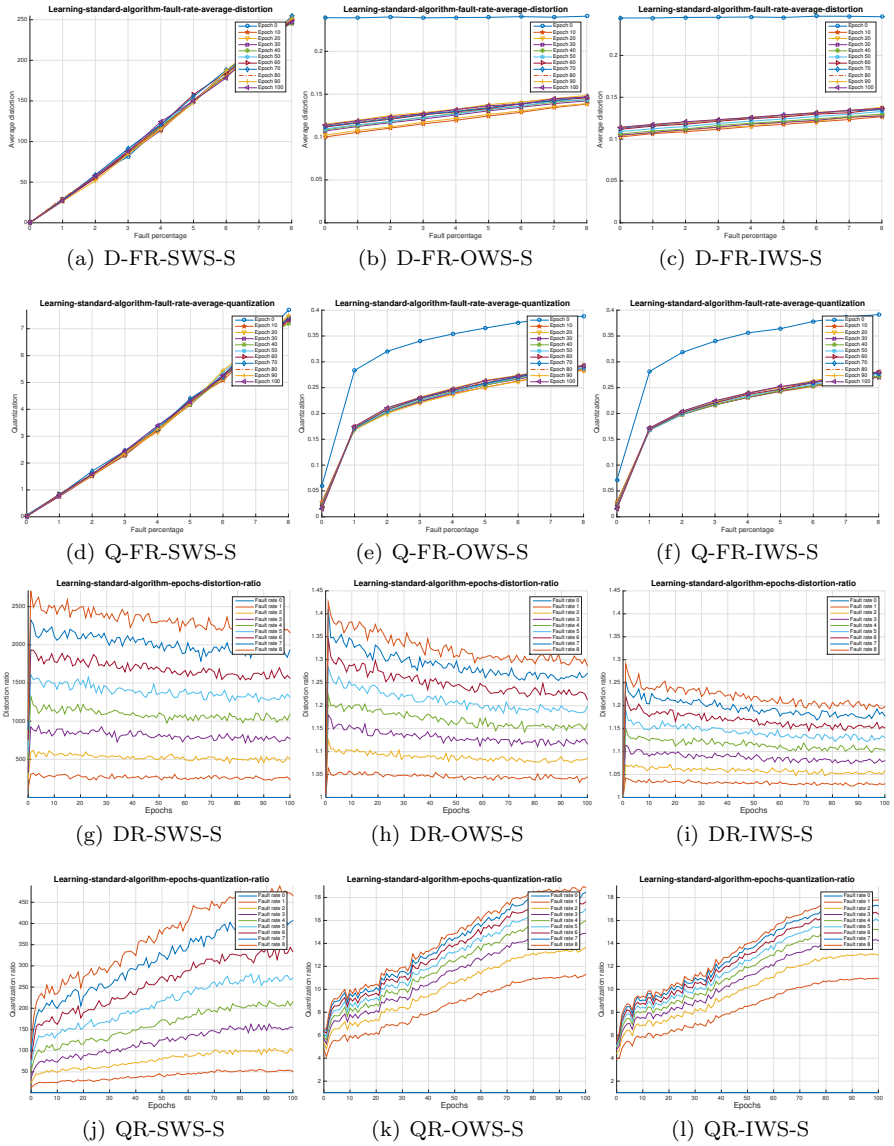
(g) DR-SWS (h) DR-SWS (i) DR-SWS

(j) QR-SWS (k) QR-SWS (l) QR-SWS

Fig. 8 Distortion/Quantization vs fault rate for different learning techniques ((a,d,g,j):thresholding, (b,e,h,k): noise injection, (c,f,i,l): fault injection) on a 2D boxes distribution (8x8 map threshold).

for quantization measure for higher fault rates, but also for distortion as it was already shown in figures 5(a)-(c).

Plots in figure 8 show that similar results for quantization and distortion are obtained for the other learning techniques, thresholding and noise injection. This is also the case when considering IWS or OWS. However, fault injection



(a) D-FR-SWS-S (b) D-FR-OWS-S (c) D-FR-IWS-S

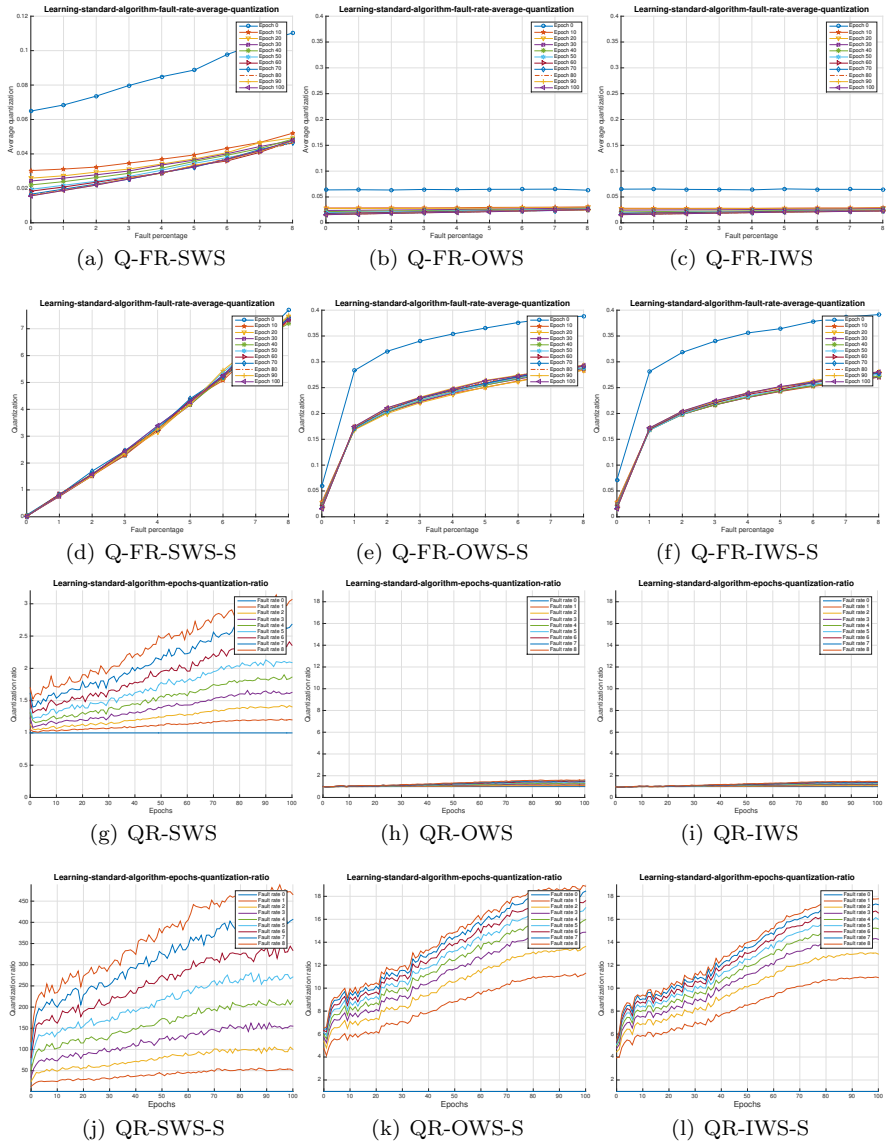
(d) Q-FR-SWS-S (e) Q-FR-OWS-S (f) Q-FR-IWS-S

(g) DR-SWS-S (h) DR-OWS-S (i) DR-IWS-S

(j) QR-SWS-S (k) QR-OWS-S (l) QR-IWS-S

Fig. 9 Distortion/Quantization vs fault rate for different implementation choices (weight storage) on a 2D boxes distribution (8x8 map sequential).

performs worst among the learning techniques, despite what can be observed in figure 8(i): the distortion ratio only decreases because the performance without faults is already bad.



(a) Q-FR-SWS (b) Q-FR-OWS (c) Q-FR-IWS

(d) Q-FR-SWS-S (e) Q-FR-OWS-S (f) Q-FR-IWS-S

(g) QR-SWS (h) QR-OWS (i) QR-IWS

(j) QR-SWS-S (k) QR-OWS-S (l) QR-IWS-S

Fig. 10 Quantization vs fault rate for different implementation choices (weight storage) on a 2D boxes distribution (8x8 map). (a-c) and (g-i): parallel, (d-f) and (j-l): sequential.

5.4 Sequential implementation results

Figure 9 shows the distortion/quantization for different arithmetic precision choices of the sequential implementation evaluated on a 2D boxes distribution. The effect on distortion is negligible, even if in average the BMU is changed by sequential faults very often (half of the time for only 2% fault rate), but

the effect on quantization is much higher. To illustrate this, figure 10 compares quantization for the parallel and sequential architectures for different arithmetic choices. It clearly shows that quantization in the sequential implementation is more affected than in the parallel one.

6 Conclusions

This paper presents a study of fault tolerance of SOMs, particularly how fault tolerance evolves during learning, assuming a hardware-plausible fault model that particularly fits FPGA implementations. We compare the performances of SOMs obtained with three variants of the on-line training algorithm: weight restriction, fault injection and noise injection. The obtained results show that training by inserting noise injection provides slightly better results than the other techniques but its results are close to the standard SOM learning algorithm. Nevertheless, SOM fault tolerance is globally confirmed since the quantization quality degrades gradually with increasing fault rates. SOM fault tolerance with respect to structure preservation has been studied using the distortion measure, and results show that this particular property of SOMs is not necessarily capable of supporting a large number of faults. An individualized weight storage policy for the FPGA implementation greatly improves this fault tolerance. The distortion measure slightly increases with the fault rate using such weight storage, corresponding to a graceful degradation of the structure of the map. The quantization measure even improves with faults, especially with large maps or during the early stages of self-organization. A study of the evolution of neural prototypes with faults and learning shows that faulty SOMs combine the partially preserved map with new prototypes that faults are spreading within the bounds of the input space thanks to the individualized weight storage policies for which faulty bits can only affect significant bits of the weights. These results confirm the intrinsic fault tolerance of self-organizing maps when they are implemented in a way that takes advantage of their parallel structure and using adequate technological choices. As illustrated by the experimental results, the distortion measure does not appear as really able to account for the structure preservation of a SOM in a precise way. In our future works, we intend to address this limit by introducing more evolved measures, in relation with the idea of population coding of the BMU (groups of neurons instead of a single winner), a paradigm for which we expect an even better fault tolerance and thanks to which SOMs may outperform other quantization models that do not order their prototypes according to a predefined topology.

References

1. Abraham, J.A., Fuchs, W.K.: Fault and error models for vlsi. Proceedings of the IEEE **74**(5), 639–654 (1986)

2. Alippi, C.: Selecting accurate, robust, and minimal feedforward neural networks. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **49**(12), 1799–1810 (2002)
3. Allende, H., Moreno, S., Rogel, C., Salas, R.: Robust self-organizing maps. In: A. Sanfeliu, J.F. Martínez Trinidad, J.A. Carrasco Ochoa (eds.) Progress in Pattern Recognition, Image Analysis and Applications: 9th Iberoamerican Congress on Pattern Recognition, CIARP 2004, Puebla, Mexico, October 26-29, 2004. Proceedings, pp. 179–186. Springer Berlin Heidelberg (2004)
4. Avižienis, A.: Framework for a taxonomy of fault-tolerance attributes in computer systems. SIGARCH Comput. Archit. News **11**(3), 16–21 (1983)
5. Beaton, D., Valova, I., MacLean, D.: Cqoco: A measure for comparative quality of coverage and organization for self-organizing maps. Neurocomputing **73**(10–12) (2010)
6. de Bodt, E., Cottrell, M., Verleysen, M.: Statistical tools to assess the reliability of self-organizing maps. Neural Networks **15**(8–9), 967 – 978 (2002)
7. Cavalieri, S., Mirabella, O.: A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks. Neural Networks **12**(1), 91 – 106 (1999)
8. Chin, C.T., Mehrotra, K., Mohan, C.K., Rankat, S.: Training techniques to obtain fault-tolerant neural networks. In: Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on, pp. 360–369 (1994)
9. Cottrell, M., Olteanu, M., Rossi, F., Villa-Vialaneix, N.: Theoretical and applied aspects of the self-organizing maps. In: Advances in Self-Organizing Maps and Learning Vector Quantization: Proc. of the 11th Int. Workshop WSOM, pp. 3–26 (2016)
10. Girau, B., Torres-Huitzil, C.: Optimal weight storage improves fault tolerance of soms. In: 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–6 (2017)
11. Hammadi, N.C., Ito, H.: Improving the performance of feedforward neural networks by noise injection into hidden neurons. Journal of Intelligent and Robotic Systems **21**(2), 103–115 (1998)
12. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault injection techniques and tools. Computer **30**(4), 75–82 (1997)
13. Kohonen, T.: The self-organizing map. Proceedings of the IEEE **78**(9), 1464–1480 (1990)
14. Kohonen, T.: The self-organizing map. Neurocomputing **21**(1–3), 1 – 6 (1998)
15. Maass, W.: Noise as a resource for computation and learning in networks of spiking neurons. Proceedings of the IEEE **102**(5), 860–880 (2014)
16. von Neumann, J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata Studies pp. 43–98 (1956)
17. Piuri, V.: Analysis of fault tolerance in artificial neural networks. Journal of Parallel and Distributed Computing **61**(1), 18 – 48 (2001)
18. Pölzlbauer, G.: Survey and comparison of quality measures for self-organizing maps. In: Proceedings of the Fifth Workshop on Data Analysis (WDA'04), pp. 67–82 (2004)
19. Protzel, P.W., Palumbo, D.L., Arras, M.K.: Performance and fault-tolerance of neural networks for optimization. IEEE Transactions on Neural Networks **4**(4), 600–614 (1993)
20. Rahimi, A., Benini, L., Gupta, R.K.: Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software. Proceedings of the IEEE **104**(7), 1410–1448 (2016)
21. Rougier, N., Boniface, Y.: Dynamic self-organising map. Neurocomputing **74**(11), 1840 – 1847 (2011)
22. Rynkiewicz, J.: Self-organizing map algorithm and distortion measure. Neural Networks **19**(6–7), 830 – 837 (2006)
23. Sejnowksi, T., Delbruck, T.: The language of the brain. Scientific American **307**, 54–59 (2012)
24. Sequin, C.H., Clay, R.D.: Fault tolerance in artificial neural networks. In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on, pp. 703–708 vol.1 (1990)
25. Talumassawatdi, R., Lursinsap, C.: Fault immunization concept for self-organizing mapping neural networks. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **09**(06), 781–790 (2001)

26. Tambara, L.A., Tonfat, J., Santos, A., Kastensmidt, F.L., Medina, N.H., Added, N., Aguiar, V.A.P., Aguirre, F., Silveira, M.A.G.: Analyzing reliability and performance trade-offs of hls-based designs in sram-based fpgas under soft errors. IEEE Transactions on Nuclear Science **64**(2), 874–881 (2017)
27. Tchernev, E.B., Mulvaney, R.G., Phatak, D.S.: Investigating the fault tolerance of neural networks. Neural Computation **17**(7), 1646–1664 (2005)
28. Torres-Huitzil, C., Girau, B.: Fault and error tolerance in neural networks: A review. IEEE Access **5**, 17,322–17,341 (2017)
29. Torres-Huitzil, C., Popovych, O., Girau, B.: Fault tolerance of self organizing maps. In: 2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM), pp. 1–8 (2017)
30. Ultsch, A.: Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. In: S.K. E. Oja (ed.) Kohonen Maps, pp. 33–46 (1999)
31. Venkatesh, S.S.: Robustness in neural computation: random graphs and sparsity. IEEE Transactions on Information Theory **38**(3), 1114–1119 (1992)
32. Wang, Z., Lee, K.H., Verma, N.: Overcoming computational errors in sensing platforms through embedded machine-learning kernels. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **23**(8), 1459–1470 (2015)
33. Wu, Y., Takatsuka, M.: Spherical self-organizing map using efficient indexed geodesic data structure. Neural Networks **19**(6–7), 900 – 910 (2006). Advances in Self Organising Maps - WSOM'05
34. Yasunaga, M., Hachiya, I., Moki, K., Kim, J.H.: Fault-tolerant self-organizing map implemented by wafer-scale integration. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **6**(2), 257–265 (1998)
35. Zeng, X., Yeung, D.S.: Sensitivity analysis of multilayer perceptron to input and weight perturbations. IEEE Transactions on Neural Networks **12**(6), 1358–1366 (2001)